Learning disentangled representations of repetitive textures using VAEs

Holt Spalding and Vladimir Porokhin
Fall 2019, Tufts University
holt.spalding@tufts.edu, vporok01@eecs.tufts.edu

0 Abstract

Image textures are often used in 3D computer graphics to describe local surface variations on 3D objects. Designing textures by hand can be a laborious task, and once an texture has been rendered, making changes to it can be quite difficult. We propose a method of automated texture synthesis using variational autoencoders (VAEs) capable of learning so-called "disentangled" feature representations of input data. Normally, VAE's encode all the features of input data into one vector of latent variables, however, disentangled representations encode various distinct aspects of the input data into separate variables. With this model, we are capable of learning compact, semantically-meaningful texture representations that can be manually tweaked and then expanded into novel textures with different properties. We've limited the scope of the paper to just the synthesis of repetitive textures, specifically images of brick walls. We evaluate the performance of both an unsupervised and semi-supervised VAE method by comparing their respective reconstruction errors and and by performing a qualitative exploration of the latent space variables learned by each over the course of training. We found the proposed semi-supervised method produced only slightly better image reconstructions compared to the unsupervised method, and the semantic meaning of the latent space variables was rather unclear. We believe this topic requires further study, to determine if more faithful reconstructions of the image data can be produced.

1 Motivation

Textures are commonly used in computer graphics to describe local surface variations on 3D objects. Deflection maps, for example, define small-scale perturbations of shape that would otherwise be too expensive to describe using geometry. Specular maps are used to demarcate reflective regions on objects. But of course the most well-known type of textures is the diffuse map that paints color on surfaces.

Regardless of the type, however, textures are difficult to design and work with. They require specialized expertise and substantial time investment in order to look convincing. Usually, textures are constructed manually from high-quality photos of materials and heavily edited to meet the requirements of their particular application. And if the needs change, it may be impossible to salvage and reuse the majority of work already done on the texture.

Procedural texture synthesis offers a solution to many limitations of conventional textures by providing an algorithmic way of constructing images. This approach cuts down on the time needed to adjust textures for each circumstance, but designing the required workflow is still very expensive and difficult.

1.1 Task

The goal of this project is to deliver a method for automated texture generation that does not have the pitfalls of procedural texture synthesis. Our solution involves using variational autoencoders to learn disentangled representations of textures that can then be modified to easily obtain novel images

meeting certain criteria. The complexity of procedural methods is completely avoided by this method because properties of the images do not have to be fully specified beforehand.

For simplicity, we focused on images of brick walls. This is a typical example of a repetitive material texture that is relatively easy to construct. We retrieved source photos for our dataset from Google Images and pre-processed the pictures to obtain uniform brick wall textures that could be used for training and testing.

1.2 Related Work

The variational autoencoder model used in this project is based on the work previously done by Siddharth et al. [1] In their paper, the authors presented a novel framework that allows variational autoencoders to be trained in a semi-supervised fashion regardless of the conditional dependency structures underlying the distribution over the encoded representations and the final decoded product. They applied that technique to the MNIST handwritten digits and "Yale B" intrinsic faces datasets, generating new and adjustable images with extremely impressive visual fidelity.

Recently (2019), Hu et al. [2] worked on the same problem, though their approach was different. They obtained a collection of user-submitted procedural synthesis graphs from a website, and then trained different convolutional neural networks to learn model-specific parameters. Since their technique uses pre-constructed procedural models, it can generate new images with little effort, however, it is strictly limited to the set of models it can choose from. It cannot construct new textures from training images alone.

1.3 Hypothesis

We hypothesize that the semi-supervised aspect of the technique from Siddharth et al. is crucial to its success. The presence of labels on at least a fraction of the training set allows their model to learn disentangled representations, whereas an unsupervised implementation would be forced to learn fully-entangled image descriptions. As a result, while the quality of training reconstructions may be the largely same between the two , the quality of novel constructions with modified latent representations would be greatly impacted for the unsupervised model. To test this theory, we have designed an experiment that applies both the semi-supervised and unsupervised variants of the technique to a collection of images and compared the results.

To evaluate the quality of reconstructions during training, we rely on binary cross entropy (BCE) between pixel intensities in the decoded images and those in the input textures. The overall entropy is calculated as a sum of all per-pixel BCE values. For results evaluation, we report the average root mean square error between pixels as well as the BCE.

2 Methods

2.1 Baseline Method

We closely followed the method and example of the approach presented by Siddharth et al. in their 2017 publication [1] and the accompanying ProbTorch [6] software package. The objective function for the unsupervised model is as follows:

$$\mathcal{L}^{\text{unsup}}(\phi, \theta) = \sum_{n=1}^{N} E_{q_\phi(\mathbf{z}|\mathbf{x}^n)}[\log p_\theta(\mathbf{x}^n|\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}^n)]$$

The encoder in this setup is constructed as three layers. The first layer takes an entire image $\mathbf{x}$ as a vector and produces a smaller vector of hidden values. The second and third layers each process that vector to generate the normal distribution parameters $\lambda_\phi(\mathbf{x})$ (i.e. mean and standard deviation) for every latent variable. The posterior $q_\phi(\mathbf{z}|\mathbf{x})$ then corresponds to the probability of observing a particular latent representation $\mathbf{z}$ in $\mathcal{N}(\lambda_\phi(\mathbf{x}))$, calculated using the reparametrization trick. Meanwhile, the prior $p(\mathbf{z})$ represents the probability of sampling that $\mathbf{z}$ from $\mathcal{N}(0, 1)$ (the distribution parameters are fixed in this case). Because these two terms are a part of the expectation

$E_{q_\phi(\mathbf{z}|\mathbf{x})}$, their combination is equivalent to the KL divergence between the prior over latent variables and actual distribution of latent variables as produced by the encoder.

The decoder consists of a two-layer neural network, where the latent representation $\mathbf{z}$ is transformed into a set of hidden values, followed by image reconstruction $\hat{\mathbf{x}}$. The likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ represents the reconstruction error, defined as the binary cross entropy between the original image $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$.

Since it is difficult to predict the conditional dependency structure behind our brick wall textures, we chose not to make any substantial alterations to the model used by Siddharth et al. for their MNIST experiment.

2.1.1 Implementation

Optimization is performed using the stochastic computation graph framework provided by ProbTorch. The client code defines nodes in this graph as needed to calculate the objective according to the description above, and then ProbTorch manages the computation. Internally, the package runs the Monte Carlo process to estimate the ELBO and uses backpropagation [8] to compute the gradient; for the actual optimization task, it relies on the Adam algorithm [7] implemented in PyTorch.

2.1.2 Tuning Strategy

This method offers a variety of parameters that can be adjusted for best performance. The images themselves are 32×32 grayscale maps, represented by 1024-element vectors of real numbers. This size allows for quick run times by keeping the size of neural networks small. It also has the additional benefit of being close to the 28×28 image size used by Siddharth et al. in their MNIST experiment, which makes their choices of parameters a reasonable starting point in our experiment also.

While there may be some interaction between different parameters, for simplicity reasons, we did not make it a significant factor in our process. Instead, we adopted the strategy of tuning each parameter individually, while keeping all others fixed – and once an "ideal" value was found, we moved on to adjusting the next parameter. For each parameter, we started with defaults and either adjusted it by a factor of two (up and down) or increased it in steps, analogous to an informal grid search.

The first adjustment that can be made is the number of hidden units in the encoder and decoder networks (both are the same). Increasing it can potentially improve the networks' recognition (encoding) and reconstruction (decoding) abilities, with an obvious increase in computational complexity. The ProbTorch default was 256, we found that doubling this amount twice, to $\boxed{1024}$ hidden units, performed the best without making the run time unreasonable.

The next important parameter is the number of latent variables. The dimensionality of the latent space is an information bottleneck in this design. Larger values should allow for increased level of detail, while at the same time increasing the opportunities for overfitting. Smaller values have the opposite effect. By default, ProbTorch uses 50, we also tried 100, 20, 15, 10, 9, 8, 7. $\boxed{10}$ latent variables appeared to work best.

The algorithm operates on images in batches and performs all optimization-related calculations only once per batch. As a result, this parameter can have a dramatic effect on its performance, both in terms of run time and quality of results. Insufficient batch sizes have the potential of getting "stuck" in local optima on ELBO because the optimization steps are too small. Excessive sizes may lead to a different problem, where the optimization is unable to converge on any solution. Because batch size influences convergence speed, we substantially increased the limit on number of epochs from the default 200 while testing different batch sizes. The default was 128; we tested 1, 5, 10, 20, 40, 80, 160, and found that $\boxed{80}$ images per batch offered best results.

A threshold on the number of epochs is a valid stopping condition, but it does not directly measure convergence, which was our goal. To solve this problem, we added a threshold on the change in ELBO across one epoch. In our experiments, ELBO was usually close to -10 and slowly progressed towards 0. Given that range for ELBO, we chose $\boxed{10\times^{-6}}$ as the convergence threshold, several orders of magnitude lower. This value appeared to be sufficient based on visual inspection of ELBO-over-time graphs and led to a reasonable execution time under our $\boxed{2000}$ limit on epochs.

2.2 Focus Method

The focus method adds a semi-supervised component to the training process. The objective function is based on that of the unsupervised setup, with an additional semi-supervised term pertaining to the supervised portion of the data:

$$\mathcal{L}(\phi,\theta) = \mathcal{L}^{\text{unsup}}(\phi,\theta) + \sum_{m=1}^{M} E_{q_\phi(\mathbf{z}|\mathbf{x}^m,\mathbf{y}^m)}\left[\log \frac{p_\theta(\mathbf{x}^m|\mathbf{y}^m,\mathbf{z})p(\mathbf{y})p(\mathbf{z})}{q_{\phi_\mathbf{z}}(\mathbf{z}|\mathbf{y}^m,\mathbf{x}^m)q_{\phi_\mathbf{y}}(\mathbf{y}^m|\mathbf{x}^m)}\right] + \alpha \log q_\phi(\mathbf{y}^m|\mathbf{x}^m)$$

The structure of the model is similar to that of the baseline method. In the encoder, there is a neural network layer that converts an image represented by the pixel vector $\mathbf{x}$ into a set of hidden values. Then there are two layers that process this vector of hidden values, concatenated with the supervised labels, into the normal distribution parameters $\lambda_\phi(\mathbf{x},\mathbf{y})$ – mean and standard deviation, respectively. However, there is an additional fourth layer that converts hidden values into the logit parameters $\lambda_{\phi_\mathbf{z}}(\mathbf{x})$ for the Gumbel-softmax distribution over discrete supervised labels.

The denominator in the fraction under the logarithm is a factorized representation of the encoder, designed to disentangle supervised labels $\mathbf{y}$ from unsupervised latent variables $\mathbf{z}$. The conditional probability $q_{\phi_\mathbf{z}}(\mathbf{z}|\mathbf{y}^m,\mathbf{x}^m)$ is equal to the likelihood of observing a particular latent representation $\mathbf{z}$ in $\mathcal{N}(\lambda_\phi(\mathbf{x},\mathbf{y}))$. The remaining portion of the factorization corresponds to the likelihood of observing label $\mathbf{y}$ in the discrete distribution parametrized by logits $\lambda_{\phi_\mathbf{z}}(\mathbf{x})$ and fixed temperature of 0.66.

The decoder structure is largely the same as in the baseline. It consists of a two-layer neural network that transforms the latent representation $\mathbf{z}$, alongside the supervised label $\mathbf{y}$, into a set of hidden values, which are then transformed into an image reconstruction $\hat{\mathbf{x}}$.

The numerator in the fraction under the logarithm is a factorized version of the decoder. The likelihood $p_\theta(\mathbf{x}^m|\mathbf{y}^m,\mathbf{z})$ corresponds to the probability of sampling pixel vector $\mathbf{x}$ from $\mathcal{N}(0,1)$. $p(\mathbf{y})$ is the probability of observing discrete label $\mathbf{y}$ in a Gumbel distribution where all logits are set to 0 and the temperature is set to 0.66. And lastly, the prior probability $p(\mathbf{z})$ corresponds to the likelihood of observing latent variables $\mathbf{z}$ in $\mathcal{N}(0,1)$.

2.2.1 Implementation

Aside from the technicalities of incorporating additional stochastic computation graph nodes and handling the supervised labels, there are few differences in the implementation between this method and the baseline. The algorithms and tricks used are exactly the same, with one exception. Because the supervised labels are discrete, it is impossible to use normal Gaussian distribution or the like to describe them in a differentiable way. The Gumbel-softmax trick works around this problem.

2.2.2 Tuning Strategy

This technique had the same configurable parameters as the baseline unsupervised method, with the exception of the percentage of images that had supervised labels associated with them. However, because this step merely determined how the data was filtered and loaded, and had no effect on the algorithm otherwise, we opted to keep it at the default $\boxed{10\%}$ split.

The other parameters were not changed from the baseline in order to ensure the results are comparable.

2.3 Data Preparation

We collected 50 high-definition photos of brick walls from Google Images, all taken with a consistent front perspective. The images were downscaled and divided to produce a collection of 32×32 unique tiles. In addition, all tiles were rotated by -60, -45, -30, 30, 45, 60, and 90 degrees and labeled accordingly. To avoid introducing empty spots around the edges due to rotation, the tiles were cropped twice: first as 32×32 blocks with extra $\lceil 32\sqrt{2}/2 \rceil$ pixels away from each side, then rotated, then cropped again to their final 32×32 dimensions. "Overscanning" the tile during the first crop ensures that no empty spots will appear in the final image under the worst case scenario of 45 degree rotation. Finally, after minor contrast enhancement, pixel brightness was adjusted on per-block basis to keep the overall exposure consistent.

To aid with testing and avoid using lower quality photos, we created two completely independent datasets from a subset of images downloaded earlier. The first set contains 603 tiles per rotation angle and was based on 9 different photos. The second set was smaller, with 175 tiles per class derived from 4 images.

Alternatives to designing our own dataset were considered, including the Describable Texture Dataset [3], the Brodatz dataset [4], and Trunk12 [5]. Unfortunately, none of them were able to afford the quality and quantity we needed for this experiment.

3 Experiments

In our experiments, we ran the two VAE methods on the dataset consisting of rotated images. The goal was to show that the semi-supervised technique is capable of achieving superior image reconstruction quality because it can learn disentangled – and thus more meaningful and compact – representations. The images were rotated during early pre-processing stages, and were already labeled by their orientation angle. Once a model was trained on the large 603-tile set, it was applied to the 175-tile set to evaluate its ability to generalize its training to image it has never seen before.

In Figure 1, we provide a visual comparison between the two techniques. Although the reconstructions are not of production quality, interesting patterns emerge from this experiment. Firstly, both methods appear to learn the grout lines between brick layers with relative ease, the short lines separating individual bricks, however, proved to be a significant challenge for them. Also, the quality of reproductions sharply dropped when the methods were switched over to the test set. This discrepancy is a strong indicator of model overfitting, as the VAEs failed to generalize their training to new images. This qualitative evaluation implies that the semi-supervised nature of the focus method does not inherently improve reconstructions.

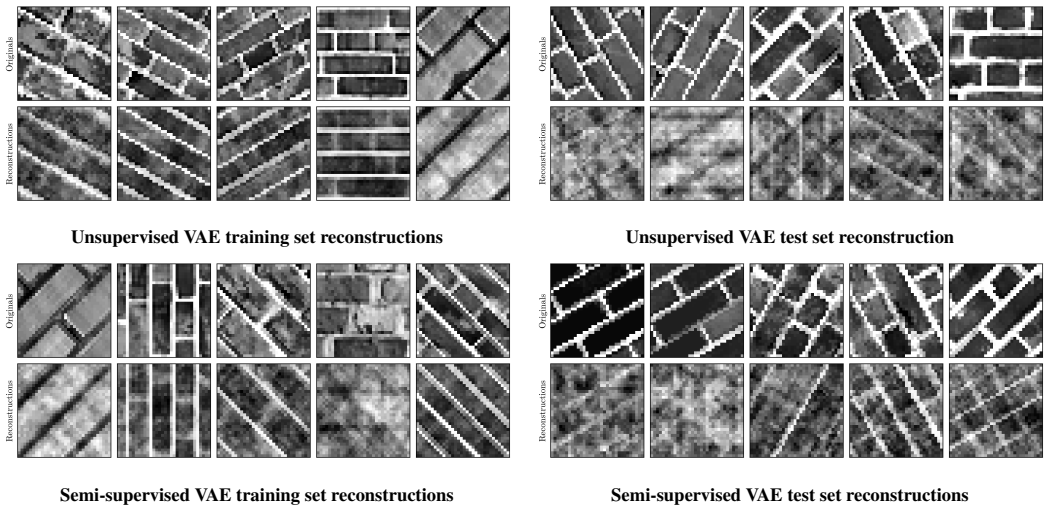
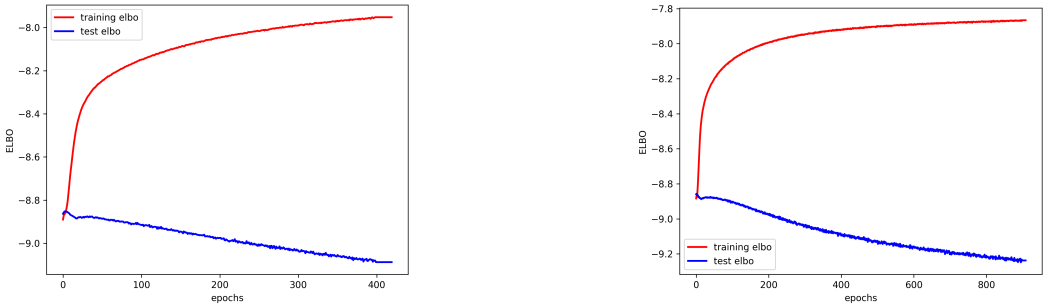


Figure 1: Comparison of 5 random image reconstructions for the training (left) and test sets (right), performed by unsupervised (top) and semi-supervised (bottom) VAE techniques.

Looking at the evolution of ELBO over the epochs (Figure 2), the tendency of both methods towards overfitting becomes very apparent. As the VAE progressed, it had no difficulty improving the objective evaluated on the training set. However, at the same time the ELBO computed over the test set was steadily going down. In other words, the VAE learned to optimize the objective function at the expense of generalizability, and the semi-supervised aspect made no difference.

Experiment	Epoch	Training BCE	Test BCE	Training RMSE	Test RMSE
Baseline	0	-708.68	-708.651	0.213	0.213
	422	-676.148	-676.116	0.174	0.174
Focus	0	-707.268	-707.270	0.211	0.211
	909	-668.833	-668.868	0.164	0.164

Table 1: Binary Cross Entropy and Root Mean Square Error for each of the methods, at the start of the algorithm, and after convergence.



Unsupervised VAE

Semi-supervised VAE

Figure 2: ELBO optimization over the course of the algorithm. Red line denotes the objective computed on the training set, blue line corresponds to the same for the test set.

To obtain a quantitative measure of reconstruction quality, we calculated the average Binary Cross Entropy and Root Mean Square Error for all images in each training set at the beginning and end of the algorithm (Table 1). Within each method, we observed an improvement in both metrics as a result of training activity, suggesting that the techniques enhanced the quality of their reconstructions, at least in terms of BCE and RMSE. However, between the two methods, there was little to no difference – before or after the training.

For our last experiment, we wanted to visually evaluate the ability of semi-supervised VAE to learn texture rotation disentangled from the rest of its features. After training, we created an encoding of a random image, and then created new reconstructions by varying its label. The results can be seen in figure 3: it appears that the model has captured the correspondence between some rotation angles and the orientation of long grout lines, but it was unable to learn the transformation in the general sense.

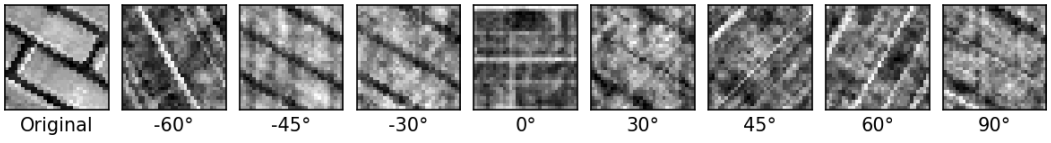


Figure 3: Image from the training set: original, alongside reconstructed with different class labels

4 Reflection and Outlook

4.1 Short-term

Based on our experiments, configuring a semi-supervised VAE model capable of learning disentangled representations is challenging in its own right. There are many parameters that can be adjusted, including the model structure itself, and due to interactions between them, it is difficult to predict the effect of any change without continuously testing the model.

We believe it will be possible to achieve a quick improvement by simply increasing the fraction of labeled data used during training. This might help with differentiating the two methods, as it will put increased emphasis on the part of the objective that is unique to the semi-supervised method.

4.2 Long-term

The most significant shortcoming of this work appears to be its reliance on the MNIST stochastic computation graph model from published literature. While the model works well in that application, the conditional dependency structure it implements may be inadequate for brick wall images. Future work can benefit from investigating other dependency structures that would make sense for the task.

Overall, VAEs appear to be a viable strategy for image construction, however, careful consideration needs to be given to the structure and model parameters before it would work. It is not a plug-and-play method, despite what other publications may imply!

References

[1] Siddharth, N. and Paige, Brooks and van de Meent, Jan-Willem and Desmaison, Alban and Goodman, Noah D. and Kohli, Pushmeet and Wood, Frank and Torr, Philip. (2017) "Learning Disentangled Representations with Semi-Supervised Deep Generative Models." *Advances in Neural Information Processing Systems 30.* Curran Associates, Inc. pp. 5927-5937

[2] Y. Hu, Dorsey, J., and Rushmeier, H., "A Novel Framework For Inverse Procedural Texture Modeling", *ACM Transactions on Graphics, vol. 38, no. 6, 2019.*

[3] Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., & Vedaldi, A. (2014). Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3606-3613).*

[4] Brodatz, P. (1966). Textures: a photographic album for artists and designers. Dover Pubns. https://multibandtexture.recherche.usherbrooke.ca/original_brodatz_more.html

[5] Fekri-Ershad, S. (2019). A New Benchmark Dataset for Texture Image Analysis and Surface Defect Detection. arXiv preprint arXiv:1906.11561.

[6] Jan-Willem van de Meent, Siddharth Narayanaswamy, Brooks Paige, Alban Desmaison, Alican Bozkurt, Amirsina Torfi, Babak Esmaeli, Eli Sennesh. "Probabilistic Torch." *Github.* https://github.com/probtorch/probtorch/

[7] Kingma, D., Ba, J. (2014). "Adam: A Method for Stochastic Optimization." *arXiv:1412.6980v9 [cs.LG]*

[8] Rezende, D.J., Mohamed, S., Wierstra, D. (2014) "Stochastic Backpropagation and Approximate Inference in Deep Generative Models." In *Proceedings of The 31st International Conference on Machine Learning, pages 1278–1286*